Active Learning of Sequential Transducers with Side Information About the Domain

Raphaël Berthon[1,2], Adrien Boiret[1](✉), Guillermo A. Pérez[2], and Jean-François Raskin[1]

[1] Université libre de Bruxelles, Brussels, Belgium
adrien.boiret@ulb.be

[2] University of Antwerp – Flanders Make, Antwerp, Belgium

Abstract. Active learning is a setting in which a student queries a teacher, through membership and equivalence queries, in order to learn a language. Performance on these algorithms is often measured in the number of queries required to learn a target, with an emphasis on costly equivalence queries. In graybox learning, the learning process is accelerated by foreknowledge of some information on the target. Here, we consider graybox active learning of subsequential string transducers, where a regular overapproximation of the domain is known by the student. We show that there exists an algorithm to learn subsequential string transducers with a better guarantee on the required number of equivalence queries than classical active learning.

1 Introduction

Active learning is a way for a non-expert user to describe a formal object through behavioral examples and counterexamples, or to obtain formal models for the behavior of legacy or black-box systems which can subsequently be formally verified [14]. In this context, additional information about black-box systems can make learning more efficient in practice [7,13].

The L^* algorithm from [2] has been extended to learn various classes of formal object, e.g. probabilistic automata [5] and, more relevant to this paper, (subsequential deterministic) transducers on words [15]. In this work, we aim to learn transducers, and focus on a specific class of side information: an upper bound on the domain of the transduction. The advantage of this *graybox* model is twofold. First and more directly, it can be used to skip some membership queries outside the transformation's domain. Second, by looking for transducers with the proper behavior when limited to the upper bound, we allow for solutions that are smaller than the canonical objects learned by L^*. This, in turn, offers better guarantees than L^* when we consider the number of equivalence queries required to learn a target. This is relevant, as in cases like non-expert description or legacy-system

This work was supported by the ARC "Non-Zero Sum Game Graphs" project (Fédération Wallonie-Bruxelles), the EOS "Verilearn" project (F.R.S.-FNRS & FWO), and the FWO "SAILor" project (G030020N).

N. Moreira and R. Reis (Eds.): DLT 2021, LNCS 12811, pp. 54–65, 2021.
https://doi.org/10.1007/978-3-030-81508-0_5

learning, the equivalence test is realistically unreliable, or prohibitively costly, when compared to the rest of the operations.

One motivation to focus on learning transducers, and more specifically Mealy machines, with an upper bound on the domain comes from games. In multi-player verification games, assumptions about other players have been proposed to facilitate strategy synthesis [4,6, for instance]. Such assumptions also make sense when a strategy has already been obtained (via synthesis [3] or some alternative means) and one wishes to "minimize" it or its encoding. A simple way to do so is to restrict the domain of the strategy to the reachable set of game configurations (under the assumptions made about the adversaries). Finally, when the game formalism considered allows for delays or multiple choices made unilaterally by some player—as is the case in regular infinite games [8]—strategies are not implementable by Mealy machines but rather require general transducers.

Related Work. The classical algorithm for active learning is L^* [2]. It saturates a table of observations with membership queries, then building a minimal deterministic automaton compatible with those observations to send as candidate for an equivalence query. A polynomial number of membership queries and at most n equivalence queries are always sufficient to learn the automaton.

For transducers, the OSTIA algorithm [15] generalizes L^*, follows a similar structure, and offers comparable guarantees. Like in L^*, the number of queries is polynomial in the size of the minimal normal form of the target transducer.

In the case of graybox learning, the methods differ and this alters the complexity guarantees. For instance, when learning languages from so-called "inexperienced teachers" [11], one considers a case where the teacher sometimes answers a membership query with "I don't know". Under those circumstance, it is impossible to learn a unique minimal automaton. This leads to a trade-off in complexity. On the one hand, finding the minimal automaton compatible with an incomplete table of observations necessitates calls to **NP** oracles (a SAT encoding is used in [11]). On the other hand, obscuring a regular language by replacing some information with "I don't know" will always make the size of the minimal solution smaller or equal to the canonical minimal DFA.

Another work on the topic [1] concerns Mealy machines, i.e. transducers that write one letter exactly for each letter they read. It is shown that one can learn a composition of two Mealy machines if the first one is already known. Just like in [11], the L^*-type algorithm uses oracles to find minimal machines compatible with an incomplete table of observations (as we can only know the behavior of the second machine on the range of the first) and offers a guarantee in the number of equivalence queries bound to the number of states of the minimal second machine, rather than that of the composition in whole.

Contributions. We show how to use string equations that can be encoded into SAT to find a minimal transducer compatible with incomplete observations, and to use this in an L^*-like algorithm to learn transducers. Our algorithm is guaranteed to issue a number of equivalence query that is bounded by the minimal compatible transducer, rather than the canonical one. This difference can be a huge benefit when our upper bound is the result of known complex logical properties or elaborate formats respected by the input, and the transformation we wish to learn is simple.

We note the differences with [1,11] in objects learned, learning frameworks, and available queries. We focus on transducers, a class that subsumes automata and Mealy machine. As an added benefit, transducers are as compact as automata, and as or more compact than Mealy machines they are equivalent to. This compactness preserves or improves the equivalence queries guarantees. In our learning framework, the upper bound is supposed to be known by the student. This is in contrast to the inexperienced teacher case, where the scope of possible observations is unknown, and has to be assumed regular and learned on the fly. When it comes to available queries, [11] assumes the student has access to containment queries i.e. student can ask teacher if the candidates' language contains or is contained in the target, this to obtain better the guarantees. In our model, a simple equivalence query is considered. Conversely, in [1], the only way to do a membership query is to do so on the composition of both machines. In that regard, learning a composition is more constraining than learning with a domain upper bound. However, since finding a reverse image to an output word through a transducer is possible with good complexity, our algorithm can be adapted to learn a composition of two transducers, in the framework of [1].

2 Preliminaries

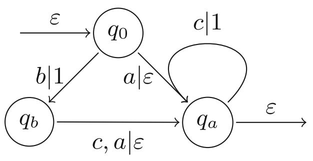
A *(subsequential string) transducer* $\mathcal{M}$ is a tuple $(\Sigma, \Gamma, Q, q_0, w_0, \delta, \delta_F)$ where Σ is the finite input alphabet, Γ is the finite output alphabet, Q is the finite set of states, $q_0 \in Q$ is the initial state, $w_0 \in \Gamma^*$ is an initial production, δ is the transition function, a partial function $Q \times \Sigma \to Q \times \Gamma^*$ and δ_F is the final function, a partial function $Q \to \Gamma^*$. If $\delta(q,a) = (q', w)$ we note $q \xrightarrow{a|w} q'$. If $\delta_F(q) = w$ we say that q is final, and note $q \xrightarrow{w} \top$. We define the relation $\to^*$ by combining the input and output of several transitions: $\to^*$ is the smallest relation such that $q \xrightarrow{\varepsilon|\varepsilon}{}^* q$, and if $q \xrightarrow{u|w}{}^* q'$ and $q' \xrightarrow{a|w'} q''$ then $q \xrightarrow{ua|w\cdot w'}{}^* q''$. We write $q_0 \xrightarrow{u|w}{}^* q$ when u reaches the state q with partial output w.

For every state $q \in Q$, we associate a partial function $[\![\mathcal{M}^q]\!](u)$ to $\mathcal{M}$ from input words over Σ to output words over Γ. Formally, $[\![\mathcal{M}^q]\!](u) = w \cdot w'$ if $q \xrightarrow{u|w}{}^* q_F$ and $q_F \xrightarrow{w'} \top$ for some $q_F \in Q$ and is undefined otherwise. Finally, we define $[\![\mathcal{M}]\!] := w_0 \cdot [\![\mathcal{M}^{q_0}]\!]$ and write that $\mathcal{M}$ implements $[\![\mathcal{M}]\!]$.

We write $\mathrm{dom}([\![\mathcal{M}]\!])$ to denote the domain of $[\![\mathcal{M}]\!]$, that is the set of all $u \in \Sigma^*$ that reach a final state $q_F \in Q$. We often consider the restriction of $[\![\mathcal{M}]\!]$ to a given domain $D \subseteq \Sigma^*$, and denote it $[\![\mathcal{M}]\!]_{|D}$.

Example 1. Consider the function τ_{abc} with domain $Up_{abc} = (a + bc)c^*$ and $\tau_{abc}(ac^n) = \tau_{abc}(bc^n) = 1^n$. It is implemented by the left transducer in Fig. 1.

We note that if we want to restrict a transducer's function to a regular language L for which we have a deterministic word automaton $\mathcal{A}$, a classic construction is to build the product transducer $\mathcal{M} \times \mathcal{A}$, where the states are the



	ε	c
$\varepsilon \in [\varepsilon]$	$\perp$	$\perp$
$a \in [a]$	ε	#
$b \in [b]$	$\perp$	1
$ac \in [a]$	#	#
$bc \in [a]$	1	#

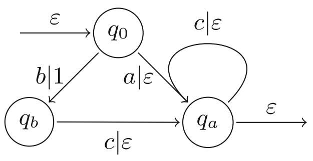


Fig. 1. On the left, a transducer compatible with the merging map in the center, on the right the transducer resulting from this merging map.

Cartesian products of both state spaces, and the final function δ_F is only defined for pairs (q, p) where q is in the domain of the final function of $\mathcal{M}$ and p is final in $\mathcal{A}$. This transducer implements the function $[\![\mathcal{M}]\!]_{\mathcal{A}}$.

We write $|\mathcal{M}|$ to denote the *size* of $\mathcal{M}$, i.e. its number of states. For convenience, we only consider *trim* transducers, that is to say that every state q is reachable from q_0 and co-reachable from a final state. This is no loss of generality, as every transducer is equivalent to a trimmed one with as many or fewer states, and we only consider minimal transducers.

Active Learning. Let Σ and Γ be finite input and output alphabets respectively. Further, let $\tau\colon \Sigma^* \to \Gamma^*$ be a partial function implementable by a transducer. In this work we will be interested in *actively learning* a transducer implementing τ by interacting with a *teacher* who knows τ and can answer questions our algorithm asks about τ. Formally, the teacher is an oracle that can answer *membership* and *equivalence* queries.

Given $u \in \Sigma^*$, a membership query answers $\tau(u)$ if $u \in \mathrm{dom}(\tau)$, and $\perp$ otherwise. Given $\mathcal{M}$ a transducer, an equivalence query answer *true* if $[\![\mathcal{M}]\!] = \tau$, otherwise it provides $u \in \Sigma^*$, a non-equivalence witness such that $u \in \mathrm{dom}([\![\mathcal{M}]\!]) \backslash \mathrm{dom}(\tau)$, or $u \in \mathrm{dom}(\tau) \backslash \mathrm{dom}([\![\mathcal{M}]\!])$, or $u \in \mathrm{dom}([\![\mathcal{M}]\!]) \cap \mathrm{dom}(\tau)$ but $[\![\mathcal{M}]\!](u) \neq \tau(u)$. The goal of a learning algorithm in this context is to produce a transducer $\mathcal{M}$ such that $[\![\mathcal{M}]\!] = \tau$.

Side Information About the Domain. We generalize the active learning problem by introducing side information available to the learning algorithm. Concretely, we assume that an *upper bound* on the domain of τ is known in advance. That is, we are given a DFA $\mathcal{A}_{Up}$ whose language Up is such that $\mathrm{dom}(\tau) \subseteq Up$. The goal of the learning algorithm is to produce a transducer $\mathcal{M}$ such that $[\![\mathcal{M}]\!]_{|Up} = \tau$.

The domain upper bound Up may allow us to learn simpler transducers $\mathcal{M}$ than the canonical minimal transducer describing τ—i.e. the class of transducers learnt by OSTIA. For instance, consider the domain Up is the set of BibTeX references where n different properties appear (title, author, year...), but in any order. The automaton recognizing this domain has $\mathcal{O}(2^n)$ states. Learning it, or any transformation on this domain, with a blackbox algorithm may thus require $\mathcal{O}(2^n)$ equivalence tests. However, if the transformation we want to learn is just to extract the title, ignoring every other property, then there exists a very simple transducer, whose size does not increase with n and that, when restricted to Up, performs the desired transformation.

3 Learning Transducers with Side Information

Our algorithm uses an *observation table* T based on a finite prefix-closed subset P of Σ^* and a finite suffix-closed subset S of Σ^*. Formally, we define T as a function $(P \cup P \cdot \Sigma) \cdot S \to \Gamma^* \cup \{\#, \perp\}$ and maintain the following invariant for all $u \in (P \cup P \cdot \Sigma)$ and all $v \in S$. If $u \cdot v \notin \mathit{Up}$ then $T(u \cdot v) = \#$. If $u \cdot v \in \mathit{Up} \setminus \mathrm{dom}(\tau)$ then $T(u \cdot v) = \perp$, otherwise $T(u \cdot v) = \tau(u \cdot v)$. For technical reasons, proper to graybox learning [11], we often consider the set P_T of prefixes of the elements of $(P \cup P\Sigma) \cdot S$.

Definition 2 (Compatible transducer). *Let T be an observation table and $\mathcal{M}$ a transducer of input alphabet Σ and output alphabet Γ. We say that $\mathcal{M}$ is compatible with T when for all $u, v \in P \cup P\Sigma$, if $T(u \cdot v) \in \Gamma^*$ then $[\![\mathcal{M}]\!](u \cdot v) = T(u, v)$ whereas if $T(u \cdot v) = \perp$ then $u \cdot v \notin dom([\![\mathcal{M}]\!])$.*

To “fill” the table so as to satisfy the invariant, we pose membership queries to the teacher. Once T has certain satisfactory properties (as defined in the OSTIA algorithm and elaborated upon briefly), we are able to construct a transducer $\mathcal{M}$ from it. As a table T can be filled with $\#$, multiple minimal transducers may be compatible with T. To minimize the number of equivalence queries posed, we will send an equivalence query only if there is a unique minimal transducer M (up to equivalence in Up) compatible with T.

Instead of searching directly for transducers, we work only with the information on how those transducers behave on P_T. We represent this information using objects we call merging maps. We show that we can characterize when there exist two competing minimal transducers with two different merging maps, or two competing minimal transducers with the same merging map. If neither is the case, then there is a unique minimal compatible transducer M, and we build it by guessing its merging map. We then pose an equivalence query to the teacher in order to determine whether $A_{\mathit{Up}} \times M$ implements the target function τ.

Satisfactory Properties. The following properties are those that allow the OSTIA algorithm [15] to work. Under these properties, we are sure that a transducer can be derived from the table T. They are defined on a specific table $T : (P \cup P\Sigma) \cdot S \to \Gamma^* \cup \{\perp\}$. Given $u \in P \cup P\Sigma$, we call $\mathrm{lcp}_T(u)$ the longest common prefix of all the $T(u \cdot v)$ in Γ^*. For $u, u' \in P \cup P\Sigma^*$, we say that $u \equiv_T u'$ iff for all $v \in S$, we have both $T(u \cdot v) = \perp \iff T(u' \cdot v) = \perp$ and if $T(u \cdot v) \in \Gamma^*$ then $\mathrm{lcp}_T(u)^{-1} T(u \cdot v) = \mathrm{lcp}_T(u')^{-1} T(u' \cdot v)$. A table T is *closed* if for all $ua \in P\Sigma$ there exists $u' \in P$ such that $ua \equiv_T u'$; $\equiv$-*consistent*, if for all $u, u' \in P$, $a \in \Sigma$ such that $ua, u'a \in P \cup P\Sigma^*$, then $u \equiv_T u' \implies ua \equiv_T u'a$; lcp-*consistent*, if for all $ua \in P \cup P\Sigma$, we have that $\mathrm{lcp}_T(u)$ is a prefix of $\mathrm{lcp}_T(ua)$.

The role of these notions in Algorithm 2 is twofold. First, it guarantees that the algorithm could, at worst, find the same transducer as the OSTIA algorithm [15] as a candidate for an equivalence query from a closed, $\equiv$-consistent, lcp-consistent table. Second, it can be seen as an efficient way to acquire information for a learner, as the set of words witnessing non-closure (resp. non-consistency) gives new elements to add to P (resp. S). We can see closed, $\equiv$-consistent, lcp-consistent tables as those that are saturated with membership queries, such that no further information can be obtained by a learner without resorting to more costly operations, e.g. an equivalence query.

Difficulties to Overcome. For any given table T there are infinitely many compatible transducers. This was already the case in automata or Mealy Machines [1,11]. However, where transducers differ, is that even when limiting ourselves to transducers with a minimal number of states, this might still be the case. Indeed, on some transitions, the output can be arbitrary (see Example 9). As a consequence, the method we will use to obtain a compatible transducer from a finite search space combines the methods of [11] with the addition of partial output information and an additional constraint on the output of transitions.

We want to obtain concomitantly an equivalence $\equiv$ on P_T that describes the set of states of the transducer and a *partial output* function $f : P_T \to \Gamma^*$ that describe which output is produced while reading an input. In the context of transducers, side information adds another restriction: a transducer can contain transitions that link together elements of P_T for which we have no output information in T. This is a problem, as the output of such transitions is arbitrary and leads to an infinite number of candidates.

We will represent the behavior of a transducer on P_T but keep only the output information that can be corroborated in T. We call $P_\Gamma \subseteq P_T$ the set of all $u \in P_T$ such that there exists $v \in \Sigma^*$ for which $T(u \cdot v) \in \Gamma^*$. We call $P_\emptyset \subseteq P_T$ the set of all $u \in P_T$ such that there is no $v \in \Sigma^*$ for which $T(u \cdot v) \in \Gamma^*$.

Definition 3 (Merging map). *Let T be an observation table. A* merging map *(MM) on T is a pair $(\equiv, f)$ where $\equiv$ is an equivalence relation on P_T, and f is a partial function from P_T to Γ^*, such that for all $u, u' \in P_T$ and $a \in \Sigma$:*

1. *P_T is a single equivalence class of $\equiv$.*
2. *If $f(ua)$ exists, then $f(u)$ exists and is a prefix of $f(ua)$.*
3. *If $T(u) \in \Gamma^*$ then $f(u)$ is a prefix of $T(u)$.*
4. *If we have that $f(u)$ exists, $u \equiv u'$ and $ua, u'a \in P_T$ then $ua \equiv u'a$. Furthermore, if $f(ua)$ exists then $f(u)^{-1}f(ua) = f(u')^{-1}f(u'a)$.*
5. *If $T(u) \in \Gamma^*$, $u \equiv u'$ then $T(u') \neq \bot$. Furthermore, if $T(u') \in \Gamma^*$ then $f(u)^{-1}T(u) = f(u')^{-1}T(u')$.*
6. *If $f(ua)$ exists, but there is no $v \in P_T$ such that $v \equiv u$, and $va \in P_\Gamma$, then $f(ua) = f(u)$.*

The intuition is that a MM $(\equiv, f)$ contains the information necessary to model the behavior on P_T of a transducer compatible with T. Rule 1 defines an equivalence class for all elements of P_T that would end up in a sink state. Rule 2 and 3 ensure that the output function f only grows with each transition and the final function respectively. Rule 4 and 5 ensure that the output value is properly defined for each transition and the final function respectively. Finally, rule 6 ensures we only keep output information from P_Γ. If such a pair (u, a) exists, we say that it is *muted*.

Every transducer $\mathcal{M}$ compatible with T has an underlying MM $(\equiv, f)$, and conversely, every MM $(\equiv, f)$ can be used to build a transducer $\mathcal{M}$ compatible with T. The size of a MM is the number of equivalence classes of $\equiv$ in $\mathrm{dom}(f)$. Below, we write q_u for the state associated with $u \in P_T$.

Definition 4 (Resulting Transducer). *Let T be an observation table and $(\equiv, f)$ a MM on T. In the transducer $\mathcal{M}$* resulting from $(\equiv, f)$ *the set of states is the set of equivalence classes of $\equiv$ in dom(f), the initial state is q_ε, the initial production is $f(\varepsilon)$, the transitions are $q_u \xrightarrow{a|f(u)^{-1}f(ua)} q_{ua}$ for $u, ua \in$ dom(f), and for each u such that $T(u) \in \Gamma^*$, we have $\delta_F(q_u) = f(u)^{-1}T(u)$.*

Definition 5 (Induced MM). *Let T be an observation table and $\mathcal{M}$ a transducer compatible with T. The MM $(\equiv, f)$* induced by the transducer $\mathcal{M}$ *is such that we have (A) $u \equiv v$ iff u and v reach the same state of $\mathcal{M}$; (B) for all $u \in P_T$, $a \in \Sigma$ such that $ua \in P_T$ reaches a state q of $\mathcal{M}$: (B.I) if there exists $v \in P_T$ such that $v \equiv u$, and $va \in P_\Gamma$, then $f(ua) = f(u) \cdot \delta(q, a)$ (B.II) and if (u, a) is muted, then $f(ua) = f(u)$.*

We note that these transformations are not one-to-one: some transducers compatible with T cannot be obtained with this method. For instance, let us consider a table full of #. Since no $T(u)$ is ever in Γ^*, there is no final state in any transducer created with this method. This is the goal of projecting the transducers' behavior on P_T: the MM induced by $\mathcal{M}$ only contains information on its behavior on P_T, and the transducer resulting from a MM is the transducer with the smallest amount of states and transitions whose behavior on P_T matches what is described in the MM.

Learning Algorithm. Our learning algorithm works as follows: (1) We build up T until it is closed and $\equiv$ and lcp-consistent. (2) If two minimal compatible transducers exist, we find them and a word u to tell them apart. We use a membership query on u and start again. (3) If only one minimal compatible transducer $\mathcal{M}$ remains, we find it. We use an equivalence query on $\mathcal{A}_{Up} \times \mathcal{M}$. Such an algorithm allows using the knowledge of Up to propose more compact candidates, as the minimal transducer compatible with a table T is always smaller than the canonical transducer that can be derived from T if we substitute $\perp$ for the #. This smaller model size leads to a better guarantee when it comes to the number of required equivalence queries. The full algorithm is in Algorithm 2. It uses the subprocedures CompetingMingGen and MinGen which we elaborate upon later.

Theorem 6. *Algorithm 2 terminates and makes a number of equivalence queries bounded by the number of states of a minimal $\mathcal{M}$ such that $[\![\mathcal{M}]\!]_{|Up} = \tau$.*

Proof (Sketch). We first assume termination and focus on the bound on equivalence queries. Note that, by construction of the tables, any minimal $\mathcal{M}$ such that $[\![\mathcal{M}]\!]_{|Up} = \tau$ is compatible with all of them. Thus, it suffices to argue that every equivalence query our algorithm poses increases the size of a minimal transducer compatible with it. For termination, it remains to bound the number

Algorithm 2 MinTransducerUp($\mathcal{A}_{Up}$)

```
Input: The DFA $\mathcal{A}_{Up}$ of an upper-bound
Output: A minimal DFA $\mathcal{M}$ such that $L = \mathcal{M} \cap \mathcal{A}_{Up}$
1: Let $P = S = \{\varepsilon\}$ and $T(P, S)$ the associated table
2: while True do                                  ▷ With $u, u' \in P$, $a \in \Sigma$, $v, v' \in S$
3:     if $(u, a, v, v')$ is a witness of non-lcp-consistency then add $av, av'$ to $S$
4:     else if $(u, u', a, v)$ is a witness of non-$\equiv$-consistency then add $av$ to $S$
5:     else if $ua$ is a witness of non-closure then add $ua$ to $P$
6:     else if $u :=$ CompetingMinGen($T(P, S)$) $\neq \emptyset$ then add $u$ and its suffixes to $S$
7:     else $\mathcal{M} :=$ MinGen($T(P, S)$)
8:         if $u$ is a non-equiv. witness for $\mathcal{A}_{Up} \times \mathcal{M}$ then add all its suffixes to $S$
9:         else return $\mathcal{M}$
```

of membership queries and calls to the subprocedures. Note that it is impossible to enumerate all n-state transducers compatible with an observation table. Termination will follow from the fact that we enumerate a finite subset of them. □

4 Merging Maps to Guess a Minimal Transducer

Algorithm 2 relies on CompetingMinGen(T) and MinGen(T) to find one or several competing transducers compatible with an observation table. This type of procedures is absent from blackbox learning algorithms, but central to graybox learning algorithm [11]. In the automata case, an oracle that guesses a minimal compatible automaton only needs to guess an equivalence relation on P_T. For transducers, we guess a function f that associates to each element of P_T an output in Γ^*. Since this is not a finite search space, we aim to restrict ourselves to a finite subspace that allows us to find one unique or two non-equivalent minimal candidates. We will limit the scope of this search with Definition 10 and 11 of *muted* and *open* transitions, to fix arbitrary outputs at ε.

To combine the two subprocedures, we characterize a necessary and sufficient condition for two possible minimal candidates to exist. This condition is tested by CompetingMinGen(T). When the minimal candidate is unique up to equivalence on Up, we use MinGen(T) to generate it, then send an equivalence query.

MinGen(T) Using MMs. Recall that there are transducers compatible with a table T that do not result from a MM on T. We will show that to implement MinGen(T) and CompetingMinGen(T), it is enough to focus on minimal MMs and to take the resulting transducers as candidates. To justify that this method provides the right result, we prove that it provides valid candidates.

Lemma 7. *Let $(f, \equiv)$ be a minimal MM on a table T and $\mathcal{M}$ its resulting transducer. Then, $\mathcal{M}$ is compatible with T.*

Among the minimal transducers compatible with T, there is one resulting from a MM. Indeed, from a transducer $\mathcal{M}$ compatible with T one can create a smaller one using the MM induced by $\mathcal{M}$ and Definition 4.

Proposition 8. *Let T be a table, $\mathcal{M}$ a transducer compatible with T. There is a transducer $\mathcal{M}'$, with as many states, compatible with T resulting from a MM.*

CompetingMinGen(T) Using MMs. While guessing a MM is enough to guess a minimal transducer, it does not provide a reliable way to decide whether two non-equivalent minimal compatible transducers exist. For the subroutine CompetingMinGen(T), we must find a way to detect whether this is the case. A natural first step is to say that if we can find minimal MMs whose resulting transducers are non-equivalent on Up, then we have found a solution to CompetingMinGen(T). Unfortunately, this condition is not necessary. Indeed, there are minimal MM induced by several non-equivalent transducers. This arises when a transition going out of the state associated to some $u \in P_T$ can have an arbitrarily defined output, because $ua \in P_\emptyset$, or $ua \notin P_T$.

Example 9. In Fig. 1, we note the special case of two transitions in the left transducer: the transition $q_a \xrightarrow{c|1} q_a$ linking $a \in P_\Gamma$ to $ac \in P_\varepsilon$, and the transition $q_b \xrightarrow{a|\varepsilon} q_a$ linking $b \in P_\Gamma$ to $ba \notin P_T$. In both cases, the transition is never used by any $u \in P_T$ such that $T(u) \in \Gamma^*$. The right transducer is also compatible with T, but the output of $q_a \xrightarrow{c|1} q_a$ is ε and $q_b \xrightarrow{a|\varepsilon} q_a$ has been deleted.

The first case, $ua \in P_\emptyset$, is the one we aimed to eliminate by erasing the output in muted pairs (u, a). We call muted transitions those whose output has to be ε in a transducer induced from a MM.

Definition 10. *Let T be a table, $(\equiv, f)$ a MM, and $\mathcal{M}$ its resulting transducer. For all $u \in P_T$, $a \in \Sigma$, (u, a) is a* muted pair *of $(\equiv, f)$, and $q_u \xrightarrow{a|\varepsilon} q_{ua}$ is a* muted transition *of $\mathcal{M}$, if $u, ua \in dom(f)$ but there is no $v \in P_T$ such that $u \equiv v$ and $va \in P_\Gamma$.*

The second case, $ua \notin P_T$, is new. We formalize this as follows: An open end is a place where a transition could be added without influencing the behavior of the resulting transducer on P_T. We fix the output of such transitions to ε.

Definition 11. *Let T be a table and $(\equiv, f)$ a MM. For all $u \in P_T$, $a \in \Sigma$, (u, a) is an* open end *of the map if there is no $v \in P_T$ s.t. $v \equiv u$ and $va \in P_T$. Let $\mathcal{M}$ be the resulting transducer of $(\equiv, f)$. We say that $\mathcal{M}'$ is an* open completion *of $(\equiv, f)$ (or of $\mathcal{M}$) if it is the transducer with at most one additional transition $u \xrightarrow{a|\varepsilon} u'$ per open end (u, a). We call such transitions* open transitions*.*

Muted and open transitions allow arbitrary output: if there exists a word $u \in Up$ that goes through a muted transition, that is sufficient to build several compatible transducers that give different outputs on u. This condition together with the existence of competing minimal MMs give a necessary, sufficient and effective, condition for CompetingMinGen(T).

Lemma 12. *Let T be an observation table, $(\equiv, f)$ a MM on T and $\mathcal{M}$ its resulting transducer. If there exists an open completion $\mathcal{M}'$ and an element $u \in Up$ such that $u \in dom([\![\mathcal{M}']\!])$ and u uses a muted or open transition in its run in $\mathcal{M}'$, then there exist competing minimal transducers compatible with T.*

Implementation: We prove that the following is a possible implementation of CompetingMinGen(T). (1) Search for two minimal MMs with non-equivalent corresponding transducers, (2) if these do not exist, search for a minimal MM and an open completion as in Lemma 12; (3) otherwise, we have a unique minimal transducer up to equivalence on Up.

Proposition 13. *Let T be a table. If there exist two minimal transducers $\mathcal{M}_1$ and $\mathcal{M}_2$ compatible with T but not equivalent on Up, one of the following exists: (i) two minimal MMs with non-equivalent resulting transducers $\mathcal{M}'_1, \mathcal{M}'_2$, or (ii) an open completion $\mathcal{M}'$ of a minimal MM compatible with T and a word $u \in dom([\![\mathcal{M}']\!]) \cap Up$ using at least one open or muted transition of $\mathcal{M}'$.*

5 Encoding into String Equations

Algorithm 2 would work as long as its subroutines return the desired results. While it is impossible to enumerate all compatible transducers, one way to find compatible transducers would be to enumerate all MM. For complexity's sake and to align our result with other graybox algorithms [1,11], we encode the minimal generation subroutines into an **NP** problem like SAT. While a direct encoding is possible, it is easier to go through a first encoding into string equations. We only use operations that are easily encoded into SAT: word (in)equality, concatenation, Boolean operators, and a restricted use of quantifiers.

This setting has the advantage of being more directly relevant to the notions we consider, while keeping the **NP** theoretical bound. Furthermore, SMT solvers have specialized tools [12,16] to solve such equations, that may yield better practical results than a direct SAT encoding.

We encode a table T, merging maps $(\equiv, f)$, and runs of $u \in Up$ with output $w \in \Gamma^*$ in the resulting transducer of T. We use word variables T_u for $T(u)$, booleans $E_{u,v}$ for $u \equiv v$, word variables f_u for $f(u)$, word variable u and letter variables $a_i \in [1, k]$ with $u = a_1 \cdot \dots \cdot a_k$ for an input word of length k, and word variables $w = w_0 \cdot w_1 \cdot \dots \cdot w_k \cdot w_{k+1}$ for their output step by step. The bounds on the size of u is given by small model theorems in automata and transducers.

We use string equation formulae to encode the properties we combine in the minimal generation subroutines. We classically build ϕ_{eq} that ensures $E_{u,v}$ denotes an equivalence. Then, each point of Definition 3 can be seen as a simple combination of string equations on T_u and f_u using the binary variables $E_{u,v}$ for $u, v \in P_T$. We can thus build ϕ_{mm} that ensures $E_{u,v}$ and f_u denote a MM.

For the transducer resulting from $(\equiv, f)$, and its open completions, we add booleans $m_{u,a}$, $o_{u,a}$ that indicate leaving q_u with a is a muted or open transition.

To model runs, we use $\phi_{run}(u, w)$ ensuring u has a run with output w in the transducer resulting from $E_{u,v}$ and f_u. We build it by making sure the run starts in the initial state with production w_0, ends in a final state with production w_{k+1}, and at the i^{th} step, the input letter is a_i and the output word is w_i.

To encode MinGen(T) we only need to find $E_{u,v}, f_u$ that respect ϕ_{mm} with n states, where n starts at 1 and increases until a solution is found.

For CompetingMinGen(T), we use Proposition 13 to split the encoding in two. To encode the case where there exist two non-equivalent MMs, we use variables $E_{u,v}$ and f_u respecting ϕ_{mm} for a first MM, copies $E'_{u,v}$ and f'_u respecting ϕ_{mm} for a second MM, and ϕ_{Up} and ϕ_{run} to encode the existence of $u \in Up$ whose run differs in the transducers resulting from both MMs.

It is easy to encode the case where there exist an open completion and a word $u \in Up$ that uses an open or muted transition, by using $m_{u,a}$ and $o_{u,a}$ on the run of u in the transducer resulting from the MM of $E_{u,v}$ and f_u.

Combined together, they encode the minimal generating subroutines in string equations, that could then be encoded in SAT, leading to our result:

Proposition 14. *Let T be an observation table. The subroutines MinGen(T) and CompetingMinGen(T) can be effectively implemented.*

Note on Complexity: As this string-equation encoding is a polynomial shorthand for a SAT encoding, each oracle call solves an **NP** problem. Coarsely, MinGen(T) and CompetingMinGen(T) are of complexity $\mathbf{P}^{\mathbf{NP}}$. To find a minimal MM of size n, we need $n-1$ of those oracles to fail on sizes $1 \leq i < n$. If we take Algorithm 2 in its entirety, each call to MinGen(T) and CompetingMinGen(T) need not make use of n oracle calls since we can cache current minimal size for future calls.

6 Conclusion

Adapting graybox learning to transducers revealed more complex than expected. Our solution relies on merging maps, muted and open transitions while offering better bounds on equivalence queries than OSTIA. Two main questions remain open: (1) The bound on the number of equivalence queries was the aim of this paper, but the number of membership queries or call to string equations solvers are not considered. Providing tight bounds or proposing a potential tradeoff, like the one described in [1], would increase the viability of the implementation of such an algorithm. (2) We could consider other classes of side information like general upper bound that cut sections of $\Sigma^* \times \Gamma^*$.

As practical future work, we plan to apply our learning algorithm to the minimization of strategies synthesized by tools participating in the *Reactive Synthesis Competition* [9]. In one of the tracks from the competition, specifications are even given in a format where assumptions about the environment are explicit [10]. We expect our algorithm to work best for that setup.

References

1. Abel, A., Reineke, J.: Gray-box learning of serial compositions of mealy machines. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 272–287. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40648-0_21
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)

3. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 921–962. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_27
4. Brenguier, R., Raskin, J., Sankur, O.: Assume-admissible synthesis. Acta Informatica **54**(1), 41–83 (2017). https://doi.org/10.1007/s00236-016-0273-2
5. de la Higuera, C., Oncina, J.: Learning stochastic finite automata. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 175–186. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30195-0_16
6. Fisman, D., Kupferman, O., Lustig, Y.: Rational synthesis. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 190–204. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_16
7. Garhewal, B., Vaandrager, F., Howar, F., Schrijvers, T., Lenaerts, T., Smits, R.: Grey-box learning of register automata. In: Dongol, B., Troubitsyna, E. (eds.) IFM 2020. LNCS, vol. 12546, pp. 22–40. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63461-2_2
8. Holtmann, M., Kaiser, L., Thomas, W.: Degrees of lookahead in regular infinite games. Log. Methods Comput. Sci. **8**(3) (2012). https://doi.org/10.2168/LMCS-8(3:24)2012
9. Jacobs, S., et al.: The 4th reactive synthesis competition (SYNTCOMP 2017): benchmarks, participants & results. In: Fisman, D., Jacobs, S. (eds.) Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22 July 2017. EPTCS, vol. 260, pp. 116–143 (2017). https://doi.org/10.4204/EPTCS.260.10
10. Jacobs, S., Klein, F., Schirmer, S.: A high-level LTL synthesis format: TLSF v1.1. In: Piskac, R., Dimitrova, R. (eds.) Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, 17–18 July 2016. EPTCS, vol. 229, pp. 112–132 (2016). https://doi.org/10.4204/EPTCS.229.10
11. Leucker, M., Neider, D.: Learning minimal deterministic automata from inexperienced teachers. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 524–538. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_39
12. Liang, T., Reynolds, A., Tsiskaridze, N., Tinelli, C., Barrett, C., Deters, M.: An efficient SMT solver for string constraints. FMSD **48**(3), 206–234 (2016)
13. Neider, D., Smetsers, R., Vaandrager, F., Kuppens, H.: Benchmarks for automata learning and conformance testing. In: Margaria, T., Graf, S., Larsen, K.G. (eds.) Models, Mindsets, Meta: The What, the How, and the Why Not? LNCS, vol. 11200, pp. 390–416. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22348-9_23
14. Vaandrager, F.W.: Model learning. Commun. ACM **60**(2), 86–95 (2017). https://doi.org/10.1145/2967606
15. Vilar, J.M.: Query learning of subsequential transducers. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 72–83. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0033343
16. Zheng, Y., Zhang, X., Ganesh, V.: Z3-str: a z3-based string solver for web application analysis. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 114–124 (2013)